ISC Passive DNS

Robert Edmonds
Internet Systems Consortium, Inc.

Introduction

- Passive DNS
- SIE
- NMSG
- dnsqr
- nmsg-dns-cache
- DNSDB

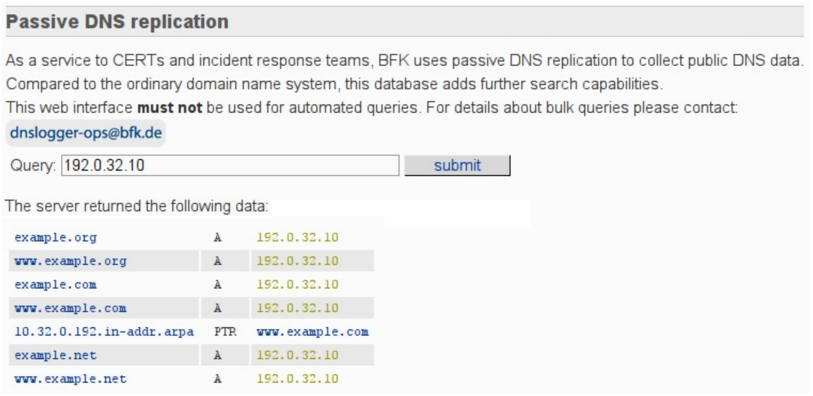
Passive DNS

- Passive DNS replication is a technology invented in 2004 by Florian Weimer.
  - Many uses: malware, e-crime, legitimate Internet services all use the DNS.
- Inter-server DNS messages are captured by sensors and forwarded to a collection point for analysis.
- After being processed, individual DNS records are stored in a database.

Passive DNS deployments

- Florian Weimer's original `dnslogger`, first at RUS-CERT, then at BFK.de (2004–).
- Bojan Zdrnja's `dnsparse` (2006–).
- ISC Security Information Exchange (2007–).

Passive DNS example #1: dnslogger



Passive DNS replication

As a service to CERTs and incident response teams, BFK uses passive DNS replication to collect public DNS data. Compared to the ordinary domain name system, this database adds further search capabilities.

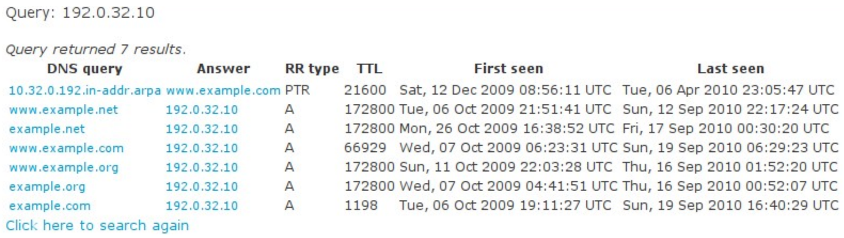
This web interface **must not** be used for automated queries. For details about bulk queries please contact: dnslogger-ops@bfk.de

Query: 192.0.32.10 submit

The server returned the following data:

example.org	A	192.0.32.10
www.example.org	A	192.0.32.10
example.com	A	192.0.32.10
www.example.com	A	192.0.32.10
10.32.0.192.in-addr.arpa	PTR	www.example.com
example.net	A	192.0.32.10
www.example.net	A	192.0.32.10

Passive DNS example #2: DNSParse

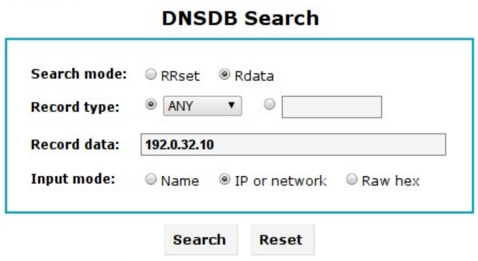

Query: 192.0.32.10

Query returned 7 results.

DNS query	Answer	RR type	TTL	First seen	Last seen
10.32.0.192.in-addr.arpa	www.example.com	PTR	21600	Sat, 12 Dec 2009 08:56:11 UTC	Tue, 06 Apr 2010 23:05:47 UTC
www.example.net	192.0.32.10	A	172800	Tue, 06 Oct 2009 21:51:41 UTC	Sun, 12 Sep 2010 22:17:24 UTC
example.net	192.0.32.10	A	172800	Mon, 26 Oct 2009 16:38:52 UTC	Fri, 17 Sep 2010 00:30:20 UTC
www.example.com	192.0.32.10	A	66929	Wed, 07 Oct 2009 06:23:31 UTC	Sun, 19 Sep 2010 06:29:23 UTC
www.example.org	192.0.32.10	A	172800	Sun, 11 Oct 2009 22:03:28 UTC	Thu, 16 Sep 2010 01:52:20 UTC
example.org	192.0.32.10	A	172800	Wed, 07 Oct 2009 04:41:51 UTC	Thu, 16 Sep 2010 00:52:07 UTC
example.com	192.0.32.10	A	1198	Tue, 06 Oct 2009 19:11:27 UTC	Sun, 19 Sep 2010 16:40:29 UTC

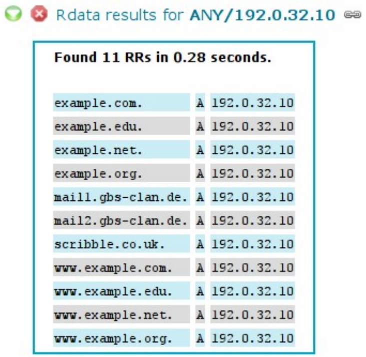
Click here to search again

Passive DNS example #3: ISC DNSDB



DNSDB Search

Search mode:	RRset [✓ Rdata]
Record type:	[✓ ANY]
Record data:	192.0.32.10
Input mode:	Name [✓ IP or network] Raw hex



Rdata results for ANY/192.0.32.10

Found 11 RRs in 0.28 seconds.

example.com.	A	192.0.32.10
example.edu.	A	192.0.32.10
example.net.	A	192.0.32.10
example.org.	A	192.0.32.10
mail1.gbs-clan.de.	A	192.0.32.10
mail2.gbs-clan.de.	A	192.0.32.10
scribble.co.uk.	A	192.0.32.10
www.example.com.	A	192.0.32.10
www.example.edu.	A	192.0.32.10
www.example.net.	A	192.0.32.10
www.example.org.	A	192.0.32.10

Passive DNS example #3: ISC DNSDB API

```
$ DNSDB_FORMAT=json isc-dnsdb-query rdata ip 192.0.32.10 | sort
{"rrtype": "A", "rrname": "example.com.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "example.edu.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "example.net.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "example.org.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "mal1.gbs-clan.de.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "mail2.gbs-clan.de.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "scribble.co.uk.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "www.example.com.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "www.example.edu.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "www.example.net.", "rdata": "192.0.32.10"}
{"rrtype": "A", "rrname": "www.example.org.", "rdata": "192.0.32.10"}
$
```

Passive DNS security issues

- **Goal: make passive DNS *at least* as reliable as DNS.**
- Passive DNS must capture both signed and unsigned data, so DNSSEC cannot help us distinguish between “good” and “bad” data.
- Record injection.
- Response spoofing.

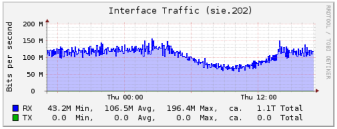
ISC Security Information Exchange

- SIE is a distribution network for different types of security data.
  - One of those types of data is passive DNS.
- Sensor operators upload batches of data to SIE.
- Data is broadcast onto private VLANs.
- `NMSG` format is used to encapsulate data.

Passive DNS channel

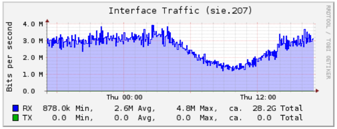
- SIE offers a “raw passive DNS” channel on VLAN 202 and further derivative channels on VLANs 204, 207.
- Some SIE passive DNS metrics based on one week of data from 2010/09/13 – 2010/09/17.
  - 2.7 billion DNS response messages per day.
  - 7.3 billion RRsets per day.
  - 100 sensors uploaded 380 GB per day, resulting in 36 Mbps of traffic.
- One-year **projections** based on one-week sample.
  - **0.98 trillion** DNS response messages per year.
  - **2.6 trillion** RRsets per year.
  - **140 terabytes** of uploads per year.

Raw passive DNS – VLAN 202 – 100 Mbps.



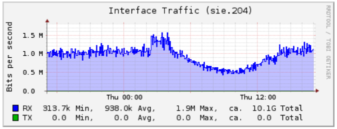
```
■ RX   43.2M Min,  106.5M Avg,  196.4M Max,  ca.   1.1T Total
■ TX    0.0  Min,     0.0 Avg,     0.0 Max,  ca.   0.0  Total
```

First stage reduction – VLAN 207 – 3 Mbps.



```
■ RX  878.0k Min,    2.6M Avg,    4.8M Max,  ca.  28.2G Total
■ TX    0.0  Min,     0.0 Avg,     0.0 Max,  ca.   0.0  Total
```

Second stage reduction – VLAN 204 – 1 Mbps.



```
■ RX  313.7k Min,  938.0k Avg,    1.9M Max,  ca.  10.1G Total
■ TX    0.0  Min,     0.0 Avg,     0.0 Max,  ca.   0.0  Total
```

NMSG

NMSG is a file and wire format for storing and transmitting blobs of information.

- Blobs of information on the order of 10 - 10,000 octets long.
- Network transport optimized for UDP over jumbo frame Ethernet.
- Framing encoded using Google Protocol Buffers. Blobs typically encoded using GPB as well.
- Different message types supported through plugin system.

Framing

- Small payloads can be coalesced into a single packet.
- Large payloads can be split across multiple packets.
- Transparent zlib compression can be applied.

Payload header

- Vendor ID
- Message type
- Timestamp
- Optional classification
  - Source
  - Operator
  - Group

Payload

- The ‘blob’ of information.
- Each (*vendor ID*, *message type*) tuple identifies a unique type of message.
- Payload blobs can be encoded with GPB, but are not required to be.

`libnmsg` message module interface

- Plugins can extend at run-time the messages types that `libnmsg` understands.
- Typically just code generated from a schema plus glue code.
    - Can be more complex: the `ISC/dns` and `SIE/dnsdedupe` message types use hooks to pretty print the data fields used in DNS.

dnsqr

- `dnsqr` is a message module for `libnmsg` specifically designed for passive DNS capture.
- Keeps track of outgoing queries and incoming responses.
- Optional filtering based on `RD` bit and `qname`.
- Structured NMSG output allows for rich programmatic access.
  - Individual fields can be accessed (`qname`, `qtype`, `rcode`, etc.)
  - Query/response messages can be retrieved.
  - Raw IP packets that contain query/response messages can be retrieved.

UDP DNS transactions

- UDP DNS packets are captured off the wire.
- UDP DNS transactions are classified into three categories:
  1. `UDP_QUERY_RESPONSE`
  2. `UDP_UNANSWERED_QUERY`
  3. `UDP_UNSOLICITED_RESPONSE`
- TODO: verify UDP checksums on responses.

```
$ isc-dnsdb-query rrset spamhaus.org/ns/spamhaus.org
;;  bailiwick: spamhaus.org.
;; first seen: 2010-06-24 03:10:20 -0000
;;  last seen: 2010-09-23 05:10:37 -0000
spamhaus.org. IN NS ns20.ja.net.
spamhaus.org. IN NS ns3.xs4all.nl.
spamhaus.org. IN NS ns3.surfnet.nl.
spamhaus.org. IN NS ns.dns-oarc.net.
spamhaus.org. IN NS ns2.spamhaus.org.
spamhaus.org. IN NS ns3.spamhaus.org.
spamhaus.org. IN NS ns8.spamhaus.org.

;;  bailiwick: spamhaus.org.
;; first seen: 2010-09-02 10:51:33 -0000
;;  last seen: 2010-09-02 10:51:33 -0000
spamhaus.org. IN NS nsr0.ja.net.
spamhaus.org. IN NS ns3.xs4all.nl.
spamhaus.org. IN NS ns3.surfn%t.nl.
spamhaus.org. IN NS ns.dns-oarc.net.
spamhaus.org. IN NS ns2.spamhaus.org.
spamhaus.org. IN NS ns3.spamhaus.org.
spamhaus.org. IN NS ns8.spamhaus.org.

;;; found 2 RRsets in 0.01 seconds
;;; SIE DNSDB
```

TCP DNS transactions

- TCP DNS packets are captured off the wire.
- TODO: perform TCP reassembly.

ICMP DNS

- ICMP packets are captured off the wire.
- Only ICMP messages that appear to be in response to a DNS message (DNS backscatter) are saved.

IP fragmentation and reassembly

- Performs IP reassembly, too.
- `nmsg/examples/nmsg-dnsqr2pcap`

State table

- The `dnsqr` state table is keyed on the DNS message 9-tuple.
  1. Initiator IP address
  2. Initiator port
  3. Target IP address
  4. Target port
  5. Internet protocol
  6. DNS ID
  7. Query name
  8. Query type
  9. Query class

Example output: `tcpdump`

```
$ tcpdump -nr iscorg.pcap
reading from file iscorg.pcap, link-type EN10MB (Ethernet)
20:55:52.970071 IP 216.27.162.20.43725 > 199.19.56.1.53: 2332 A? www.isc.org. (29)
20:55:53.214596 IP 199.19.56.1.53 > 216.27.162.20.43725: 2332- 0/4/6 (260)
```

Example output: dnsqr

```
[420] [2010-09-16 00:55:52.970071000] [1:9 ISC dnsqr]
type: UDP_QUERY_RESPONSE
query_ip: 216.27.162.20
response_ip: 199.19.56.1
proto: UDP (17)
query_port: 43725
response_port: 53
id: 2332
qname: www.isc.org.
qclass: IN (1)
qtype: A (1)
rcode: NOERROR (0)
query: [29 octets]
response: [260 octets]
```

Example output: dnsqr

```
[...]
query: [29 octets]
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 2332
;; flags:; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.isc.org. IN A

;; ANSWER SECTION:

;; AUTHORITY SECTION:

;; ADDITIONAL SECTION:
[...]
```

Example output: dnsqr

```
[...]
response: [260 octets]
;; ->>HEADER<<- opcode: QUERY, rcode: NOERROR, id: 2332
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 6

;; QUESTION SECTION:
;www.isc.org. IN A

;; ANSWER SECTION:

;; AUTHORITY SECTION:
isc.org. 86400 IN NS sfba.sns-pb.isc.org.
isc.org. 86400 IN NS ord.sns-pb.isc.org.
isc.org. 86400 IN NS ns.isc.afilias-nst.info.
isc.org. 86400 IN NS ams.sns-pb.isc.org.

;; ADDITIONAL SECTION:
ams.sns-pb.isc.org. 86400 IN A 199.6.1.30
ams.sns-pb.isc.org. 86400 IN AAAA 2001:500:60::30
ord.sns-pb.isc.org. 86400 IN A 199.6.0.30
ord.sns-pb.isc.org. 86400 IN AAAA 2001:500:71::30
sfba.sns-pb.isc.org. 86400 IN A 149.20.64.3
sfba.sns-pb.isc.org. 86400 IN AAAA 2001:4f8:0:2::19
```

sie-dns-sensor and sie-scripts

- `sie-dns-sensor` is a standalone binary distribution (`deb`/`rpm`) of `dnsqr` to aid in deployment of passive DNS sensors on Linux systems.
- `sie-scripts` is a tarball of scripts and FreeBSD ports for deploying `dnsqr`.
- Available from ftp://ftp.isc.org/isc/nmsg/misc/

nmsg-dns-cache

- `nmsg-dns-cache` is a passive DNS deduplication and filtering tool.
- Consumes raw passive DNS data from SIE channel 202.
- Emits various types of data on SIE channels 204, 206, and 207.
- Architectural split between front-end and back-end caches.

Front-end cache

- Deserialization of raw DNS messages.
- Filtering of `SOA`, `RRSIG(PTR)`s, non-`IN` class.
- RRset canonicalization.
- Filtering based on owner name.
- Insertion into and expiration from FIFO cache.
- All insertions and expirations are exported to SIE channel 207.

Front-end cache key

- The front-end deduplication cache stores a serialized form of each RRset.
  - `rrname`
  - `rrtype`
  - `rrclass`
  - `rdata array`
  - `responder IP address`

SIE/dnsdedupe NMSG schema

- Front-end and back-end caches use a special `SIE/dnsdedupe` NMSG message schema.
  - `type` (`INSERTION`, `EXPIRATION`, `CHAFF`)
  - `count`
  - `time_first`, `time_last`
  - `response_ip`
  - `rrname`
  - `rrtype`
  - `rrclass`
  - `rrttl`
  - `rdata array`
  - `bailiwick`

Back-end cache

- Deserialization of `SIE/dnsdedupe` NMSG payloads.
- Bailiwick verification.
- Regex filtering. E.g.,
  - `adsl-074-165-009-078.sip.asm.bellsouth.net.`
  - `dsl027-162-001.atl1.dsl.speakeasy.net.`
- Insertion into and expiration from FIFO cache.
- All insertions and expirations are exported to SIE channel 204.

Back-end cache key

- The back-end deduplication cache stores a serialized form of each RRset.
  - `rrname`
  - `rrtype`
  - `rrclass`
  - `rdata array`
  - `bailiwick`

Passive DNS bailiwick verification

- DNS caches associate a “bailiwick” with each outgoing query.
- The cache knows what bailiwick to use, because it knows why it’s sending a particular query.
- Passive DNS doesn’t replicate the bailiwick.
- Bailiwick must be reconstructed.

Passive DNS bailiwick algorithm

- Must operate completely **passively**.
- Must provide a boolean **true** or **false** for each record.
  - "For each record name, is the response IP address a nameserver for the zone that contains or can contain this name?"
- Example: `root` nameservers can assert knowledge about **any** name.
- Example: Verisign's `gtld` servers can assert knowledge about any domain name ending in `.com` or `.net`.

Passive DNS bailiwick algorithm

- Initialize bailiwick cache with a copy of the root zone.
  - Cache starts off with knowledge of which servers serve the root and TLDs.
- Find all **potential** zones that a name could be located in.
- Check whether any of the nameservers for those zones are the nameserver that sent the response.
- Each time an `NS`, `A`, or `AAAA` record is verified by the algorithm, it is inserted into the bailiwick cache.

Passive DNS bailiwick algorithm example

Excerpt from the DNS root zone:

```
[...]
com.                IN  NS  a.gtld-servers.net.
[...]
a.gtld-servers.net. IN  A   192.5.6.30
[...]
```

Passive DNS bailiwick algorithm example

```
;; QUESTION SECTION:
;www.example.com.        IN   A

;; AUTHORITY SECTION:
example.com.          172800  IN   NS   a.iana-servers.net.
example.com.          172800  IN   NS   b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net. 172800  IN   A    192.0.34.43
b.iana-servers.net. 172800  IN   A    193.0.0.236

;; SERVER: 192.5.6.30#53(192.5.6.30)
```

Passive DNS bailiwick algorithm example

Name: `example.com.`
Server: `192.5.6.30`

- Potential zones:
  - `example.com.`
  - `com.`
  - `.`
- Zones in bailiwick cache:
  - `com.`
  - `.`
- Check: `example.com./NS`? Not found.
- Check: `com./NS`? Found 13 nameservers.
- Check: are any of them `192.5.6.30`? Yes.

Back-end cache example output

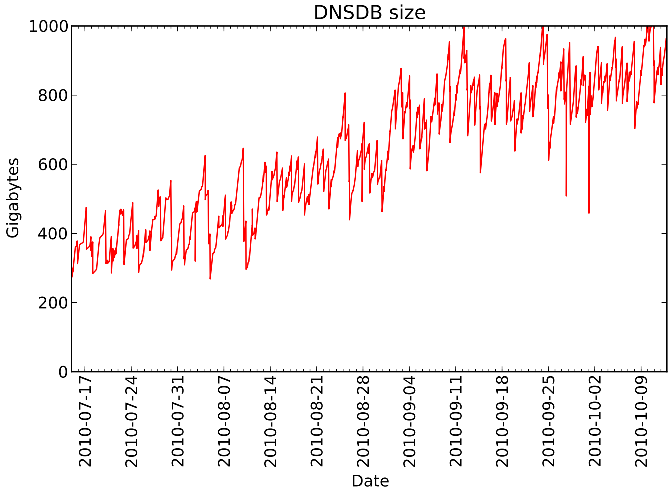
```
[134] [2010-09-22 22:41:49.509281126] [2:1 SIE dnsdedupe]
type: INSERTION
count: 1
time_first: 2010-09-22 22:40:05
time_last: 2010-09-22 22:40:05
response_ip: 216.69.185.24
bailiwick: georgesbutchershop.com.
rrname: georgesbutchershop.com.
rrclass: IN (1)
rrtype: NS (2)
rrttl: 3600
rdata: ns47.domaincontrol.com.
rdata: ns48.domaincontrol.com.
```

Back-end cache example output

```
[60] [2010-09-22 22:41:49.561219983] [2:1 SIE dnsdedupe]
type: EXPIRATION
count: 9
time_first: 2010-09-22 02:05:49
time_last: 2010-09-22 15:51:50
bailiwick: spaml.com.
rrname: www.spaml.com.
rrclass: IN (1)
rrtype: A (1)
rrttl: 1800
rdata: 64.74.223.36
```

DNSDB

- DNSDB is a database for storing DNS records.
  - Data is loaded from passive DNS and zone files.
  - Individual DNS records are stored in an Apache Cassandra database.
    - Offers key-value store distributed across multiple machines.
    - Good fit for DNS data.
    - Sustains extremely high write throughput because all writes are sequential.
  - Offers a RESTful HTTP API and web search interface.
- Database currently consumes about 900 GB out of 27 TB.


DNSDB size
Gigabytes
0
200
400
600
800
1000
Date
2010-07-17
2010-07-24
2010-07-31
2010-08-07
2010-08-14
2010-08-21
2010-08-28
2010-09-04
2010-09-11
2010-09-18
2010-09-25
2010-10-02
2010-10-09

Example

```
$ isc-dnsdb-query rrset vix.com/ns/vix.com
;;  bailiwick: vix.com.
;; first seen: 2010-06-24 03:13:44 -0000
;;  last seen: 2010-07-04 16:01:15 -0000
vix.com. IN NS ns.sjc1.vix.com.
vix.com. IN NS ns.sql1.vix.com.
vix.com. IN NS ns1.isc-sns.net.
vix.com. IN NS ns2.isc-sns.com.
vix.com. IN NS ns3.isc-sns.info.

;;  bailiwick: vix.com.
;; first seen: 2010-07-04 16:14:12 -0000
;;  last seen: 2010-09-21 12:30:57 -0000
vix.com. IN NS ns.sql1.vix.com.
vix.com. IN NS ns1.isc-sns.net.
vix.com. IN NS ns2.isc-sns.com.
vix.com. IN NS ns3.isc-sns.info.

;;; found 2 RRsets in 0.07 seconds
;;; SIE DNSDB
```

Example

```
$ isc-dnsdb-query rrset vix.com/ns/com
;;  bailiwick: com.
;; first seen: 2010-06-24 03:13:44 -0000
;;  last seen: 2010-09-21 20:31:34 -0000
;; first seen in zone file: 2010-04-24 16:12:21 -0000
;;  last seen in zone file: 2010-09-20 16:10:19 -0000
vix.com. IN NS ns.sjc1.vix.com.
vix.com. IN NS ns.sql1.vix.com.
vix.com. IN NS ns1.isc-sns.net.
vix.com. IN NS ns2.isc-sns.com.

;;; found 1 RRsets in 0.08 seconds
;;; SIE DNSDB
```

Components

- Data import.
  - Passive DNS data from SIE channel 204.
  - DNS TLD zones (FTP via ZFA programs): `com`, `net`, `org`, etc.
  - DNS zones (standard `AXFR/IXFR` protocol)
- Data retrieval.
  - HTTP API.
  - Web search interface.

Cassandra

- Cassandra is a “NoSQL” database; it doesn’t have schemas or a DDL.
- Multi-dimensional key-value store / column-oriented database.
- Allows for prefix searches of row keys.
- Writes are **fast** because all writes are sequential. Only reads require random access.
- Homepage: http://cassandra.apache.org/

Inserting data

- NMSG-formatted data is converted into Cassandra database updates and inserted in batches.
- Two Cassandra column families (or “tables”) are maintained:
  - RRset column family – stores complete RRset with metadata
  - Rdata column family – indexes Rdata to owner name / RRtype

Inserting data: RRset key

RRset “schema”: row key is the serialized form of:

- Owner name (label-reversed).
- RR type.
- Bailiwick name (label-reversed).
- Array of length-prefixed rdata values.

Inserting data: RRset key

RRset “schema”: row key is the serialized form of:

- Owner name (label-reversed).
  - **www.example.com**
- RR type.
  - **A**
- Bailiwick name (label-reversed).
  - **example.com**
- Array of length-prefixed rdata values.
  - **192.0.32.10**

Inserting data: RRset key

RRset "schema": row key is the serialized form of:

- Owner name (label-reversed).
  - **www.example.com** `\x03com\x07example\x03www\x00`
- RR type.
  - **A** `\x00\x01`
- Bailiwick name (label-reversed).
  - **example.com** `\x03com\x07example\x00`
- Array of length-prefixed rdata values.
  - **192.0.32.10** `\x00\x04\xc0\x00\x20\x0a`

Inserting data: RRset key

RRset "schema": row key is the serialized form of:

- Owner name (label-reversed).
  - **www.example.com** `\x03com\x07example\x03www\x00`
- RR type.
  - **A** `\x00\x01`
- Bailiwick name (label-reversed).
  - **example.com** `\x03com\x07example\x00`
- Array of length-prefixed rdata values.
  - **192.0.32.10** `\x00\x04\xc0\x00\x20\x0a`

Complete serialized key:

```
\x03com\x07example\x03www\x00\x00\x01\x03com\x07example\x00\x00\x04\xc0\x00\x20\x0a
```

Retrieving data: RRsets

Example: find all RRsets with the owner name `www.example.com`.

Retrieving data: RRsets

Example: find all RRsets with the owner name `www.example.com`.

Search for all keys between these key prefixes:

```
\x03com\x07example\x03www\x00
```

```
\x03com\x07example\x03www\x01
```

Retrieving data: RRsets

Example: find all RRsets with the owner name `www.example.com`.

Search for all keys between these key prefixes:

```
\x03com\x07example\x03www\x00

\x03com\x07example\x03www\x00\x00\x01\x03com\x07example\x00\x00\x04\xc0\x00\x20\x0a

\x03com\x07example\x03www\x01
```

Retrieving data: RRsets

Example: find all RRsets with the owner name `www.example.com`.

Search for all keys between these key prefixes:

```
\x03com\x07example\x03www\x00
\x03com\x07example\x03www\x00\x00\x01\x03com\x07example\x00\x00\x04\xc0\x00\x20\x0a
\x03com\x07example\x03www\x01
```

Key(s) retrieved, then deserialized.

RRset columns

Four columns currently in use.
Each column has a value (a timestamp) and another timestamp used by Cassandra for conflict resolution (higher value wins).

- “time_first” – reverse timestamp
- “time_last” – timestamp
- “zone_time_first” – reverse timestamp
- “zone_time_last” – timestamp

Timestamp: microseconds sinch epoch.
Reverse timestamp: 2^{63} minus microseconds since epoch.
Reverse timestamps used to ensure commutativity.

Inserting data: Rdata key

Rdata “schema”: row key is the serialized form of:

- Length-prefixed rdata value.
- RR type.
- Owner name (label-reversed).

Inserting data: Rdata key

Rdata “schema”: row key is the serialized form of:

- Length-prefixed rdata value.
  - **192.0.32.10**
- RR type.
  - **A**
- Owner name (label-reversed).
  - **www.example.com**

Inserting data: Rdata key

Rdata “schema”: row key is the serialized form of:

- Length-prefixed rdata value.
  - **192.0.32.10** `\x00\x04\xc0\x00\x20\x0a`
- RR type.
  - **A** `\x00\x01`
- Owner name (label-reversed).
  - **www.example.com** `\x03com\x07example\x03www\x00`

Inserting data: Rdata key

Rdata "schema": row key is the serialized form of:

- Length-prefixed rdata value.
  - **192.0.32.10** `\x00\x04\xc0\x00\x20\x0a`
- RR type.
  - **A** `\x00\x01`
- Owner name (label-reversed).
  - **www.example.com** `\x03com\x07example\x03www\x00`

Complete serialized key:

`\x00\x04\xc0\x00\x20\x0a\x00\x01\x03com\x07example\x03www\x00`

Retrieving data: Rdata

Example: find all Rdata keys that represent address records in the network `192.0.32.0/24`.

Retrieving data: Rdata

Example: find all Rdata keys that represent address records in the network `192.0.32.0/24`.

Search for all keys between these key prefixes:

`\x00\x04\xc0\x00\x20` (`192.0.32.0`)

`\x00\x04\xc0\x00\x21` (`192.0.33.0`)

Retrieving data: Rdata

Example: find all Rdata keys that represent address records in the network `192.0.32.0/24`.

Search for all keys between these key prefixes:

```
\x00\x04\xc0\x00\x20 (192.0.32.0)
\x00\x04\xc0\x00\x20\x0a\x00\x01\x03com\x07example\x03www\x00
\x00\x04\xc0\x00\x21 (192.0.33.0)
```

Retrieving data: Rdata

Example: find all Rdata keys that represent address records in the network `192.0.32.0/24`.

Search for all keys between these key prefixes:

```
\x00\x04\xc0\x00\x20 (192.0.32.0)
\x00\x04\xc0\x00\x20\x0a\x00\x01\x03com\x07example\x03www\x00
\x00\x04\xc0\x00\x21 (192.0.33.0)
```

Key(s) retrieved, then deserialized.

Rdata columns

Two columns currently in use.
Only column names used; no column values. (Column names function as booleans.)

- "p" – record observed via passive DNS replication.
- "z" – record observed in zone file.

RRsets vs Rdata

- DNS preserves RRset atomicity.
- Existing passive DNS databases do not seem to preserve RRset atomicity.
- DNSDB replicates atomic RRsets in its “rrset” column family.
- DNSDB “rdata” column family indexes individual records.
- Rdata lookups can be followed by RRset lookups to retrieve full RRset + metadata.

Software available from ftp://ftp.isc.org/isc/nmsg/:

- `nmsg`
  - `libnmsg` – NMSG C library
  - `pynmsg` – libnmsg CPython binding
  - `nmsgtool` – command line tool
  - `dnsqr` – DNS capture module
- `wreck`
  - `libwdns` – DNS message parsing C library
  - `pywdns` – libwdns CPython binding
- `sie-dns-sensor` – binary Linux packages
- `sie-scripts` – sensor scripts, FreeBSD ports

Youtube videos of previous presentations:

- "`google nmsg`" – NMSG, Google, December 2009
- "`defcon passive dns hardening`" – Passive DNS security, Defcon, July 2010

Thanks!

Questions?